US 20030131140A1

(19) United States

(12) Patent Application Publication (10) Pub. No.: US 2003/0131140 A1

Ghose et al. (43) Pub. Date: Jul. 10, 2003

(54) DATA TRANSFERS IN EMBEDDED SYSTEMS

(75) Inventors: **Arunabha Ghose**, Bangalore (IN); **Sumit Dev**, Bangalore (IN)

Correspondence Address:
TEXAS INSTRUMENTS INCORPORATED
P O BOX 655474, M/S 3999
DALLAS, TX 75265

(73) Assignee: **Texas Instruments Incorporated**, Dallas, TX (US) (US)

(21) Appl. No.: **10/256,522**

(22) Filed: **Sep. 27, 2002**

Related U.S. Application Data

(60) Provisional application No. 60/342,091, filed on Dec. 26, 2001.

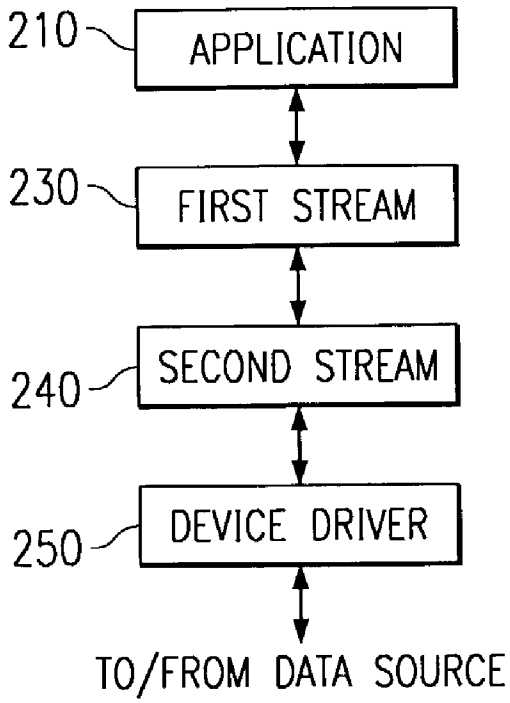
Publication Classification

(51) **Int. Cl.**[7] **G06F 15/163**; G06F 9/00

(52) **U.S. Cl.** **709/310**

(57) **ABSTRACT**

A stack (successive) of non-blocking streams may be used to transfer data from a data source to a user application. For example, one stream may transfer data from a device driver to a random access memory (RAM) and another stream may transfer the data in the RAM to an on-chip memory. By using non-blocking streams, the processing power available in an embedded system may be utilized efficiently. Another aspect of the present invention provides the user applications the ability to control the logic for allocation and release of memory space supporting buffers used in data transfers.



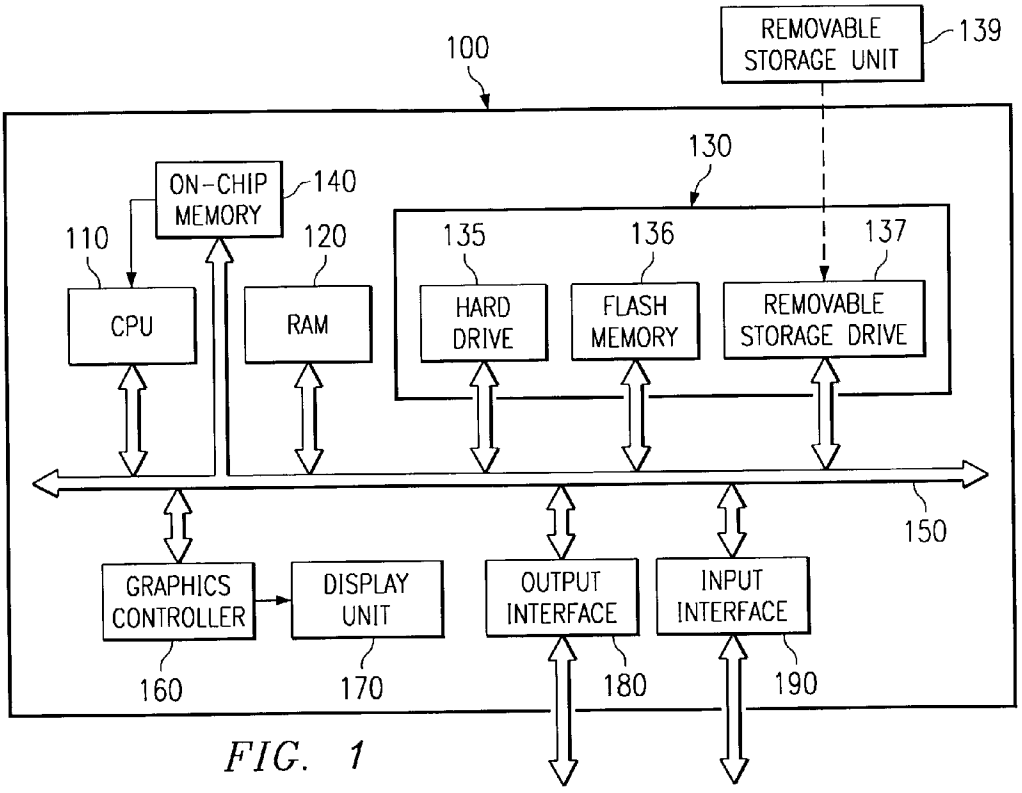

100
REMOVABLE STORAGE UNIT
139
130
ON-CHIP MEMORY
140
110
120
135
136
137
CPU
RAM
HARD DRIVE
FLASH MEMORY
REMOVABLE STORAGE DRIVE
150
GRAPHICS CONTROLLER
DISPLAY UNIT
OUTPUT INTERFACE
INPUT INTERFACE
160
170
180
190

FIG. 1

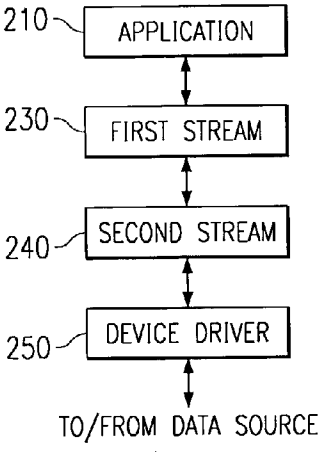

210
APPLICATION
230
FIRST STREAM
240
SECOND STREAM
250
DEVICE DRIVER
TO/FROM DATA SOURCE

FIG. 2A

FIG. 2B

```
        .
        .
        .
260  DRIVER DVR=INIT_DRIVER ()
265  DEVICE STMDEV2=INIT_DEVICE1 (DVR, ...)
266  DEVICE STMDEV1=INIT_DEVICE2 (STMDEV2, ...)
270  STREAM STM1=STMOPEN (STMDEV1,R, ...)

280  STM ISSUE (STM1, BUF, SIZE, ...)
281  .
282  .
283  .
284  STMM RECLAIM (STM1, BUFP, SIZE, ...)
```

FIG. 3

```
310  Application my-alloc
311  {
312  ...
313  }

320  Application my-free
321  {
322  ...
323  }

330  my-mem-mgr=mem-mgr
331        {my-alloc
332         my-free
333        }

350  DEVICE STMDEV3=INIT_DEVICE3 (DVR, my-mem-mgr, ...)
351  STREAM STM3=STMOPEN (STMDEV3, w, ...)
     ...
360  buf3=STMALLOC (STM3, size3)
361  /* store the data in buf3 */
     ...
370  STMISSUE (STM3, buf3, size3)
     ...
380  STMRECLAIM (STM3, bufp, size)
     ...
390  STMFREE (STM3, buf)
```

DATA TRANSFERS IN EMBEDDED SYSTEMS

BACKGROUND OF THE INVENTION

[0001] 1. Field of the Invention

[0002] The present invention relates to embedded systems, and more specifically to a software implemented approach for efficiently transferring data in embedded systems.

[0003] 2. Related Art

[0004] Embedded systems generally refer to specialized systems used to control devices such as automobiles, home and office appliances, and handheld units (e.g, cell phones). The systems typically provide a specific set of utilities (in technology community often referred to as functions) particularly designed for the specific environments in which the systems operate.

[0005] Manufacturers of embedded systems often provide interfaces using which applications can be developed customized for the specific environments. In general, it is advantageous if the interface provides flexibility to the application developers in terms of features offered, and yet is efficient in providing the corresponding utility. One common feature needed in embedded systems is transfer of data potentially from external systems.

[0006] A prior system may only provide for blocking transfers when an application requires data from an external source. That is, the execution of the application is suspended until the requested data is retrieved from the external source and available for the application. As a result, the applications may be impeded when data transfers are required.

[0007] Another prior system may partially overcome such a deficiency by providing non-blocking interfaces. That is, an application requests the transfer of data from an external source and can continue with other tasks while the requested data is retrieved. The transfer request may be made in the form of an interface call. The application later processes the retrieved data at an appropriate time. However, such prior system may provide for only blocking implementations supporting the data transfer interface call. The blocking implementations in turn may lead to inefficiencies in the use of available processing power in an embedded system, and may thus be undesirable.

[0008] Accordingly what is needed is a method and apparatus which enables applications to efficiently continue execution when data is required from external sources.

[0009] Another desirable feature in embedded systems is providing control over memory management to the application. In some prior systems, a user application can only request (and release) the amount of memory desired and a memory manager allocates the requested memory. The user application generally does not have much control over the allocation of the memory. The allocation approaches can be based on various factors such as minimization of fragmentation, throughput performance, etc. However, the provided approach may not suit the individual application(s) using the memory manager.

[0010] Accordingly what is also needed is a method and apparatus which provides additional control to user applications over allocation of memory.

SUMMARY OF THE INVENTION

[0011] According to an aspect of the present invention a stack of non-blocking streams are used in transferring data from a data source to a user application. For example, a user application may generate a issue statement to initiate retrieval of data transfer using a lower non-blocking streaming layer.

[0012] The user application may attend to other tasks and then determine whether the requested data is available (for example, using a reclaim statement). The lower non-blocking streaming layer may also operate similarly. In an embodiment, a device driver is implemented in a blocking manner to retrieve data from a data source, a first non-blocking stream may store the retrieved data in a random access memory (RAM), and another non-blocking stream may transfer the data in the RAM into an on-chip memory for use by the user application.

[0013] Due to the non-blocking implementation of the data transfer layers, the computation power available within a system may be utilized efficiently.

[0014] According to another aspect of the present invention, the user application is provided control over the logic in allocation and release (freeing) of memory space supporting buffers. The buffers are used for transferring data across different layers performing data transfers. By providing control over memory management to the user applications, the available memory may be utilized efficiently.

[0015] Further features and advantages of the invention, as well as the structure and operation of various embodiments of the invention, are described in detail below with reference to the accompanying drawings. In the drawings, like reference numbers generally indicate identical, functionally similar, and/or structurally similar elements. The drawing in which an element first appears is indicated by the leftmost digit(s) in the corresponding reference number.

BRIEF DESCRIPTION OF THE DRAWINGS

[0016] The present invention will be described with reference to the accompanying drawings, wherein:

[0017] **FIG. 1** is a block diagram of the details of a computer system illustrating an example environment in which the present invention can be implemented;

[0018] **FIG. 2A** is a diagram illustrating the manner in which a stack of non-blocking streams can be used to provide efficient data transfers to user applications;

[0019] **FIG. 2B** contains example user application code illustrating the manner in which stack of streams can be generated to transfer data from a data source; and

[0020] **FIG. 3** contains example user application code illustrating the manner in which user applications may be provided control over the memory allocation according to a feature of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0021] 1. Overview and Discussion of the Invention

[0022] According to an aspect of the present invention, a user application can make a non-blocking streaming call to

retrieve data from an external source. The non-blocking streaming call can in turn be implemented using a non-blocking stream. Thus, a stack (successive) of streams can be used to provide desired data to user applications. Due to the implementation of stack of streams, the processing power in an embedded system can be utilized efficiently while performing data transfers.

[0023] According to another feature of the present invention, a user application is provided the ability to control allocation of memory space. The entire available memory is partitioned into multiple spaces, and a different memory manager may be associated with each space. The use application can be written to interface with the different memory managers to have the necessary memory space allocated.

[0024] The invention is described with reference to example environments for illustration. It should be understood that numerous specific details, relationships, and methods are set forth to provide a full understanding of the invention. One skilled in the relevant art, however, will readily recognize that the invention can be practiced without one or more of the specific details, or with other methods, etc. In other instances, well-known structures or operations are not shown in detail to avoid obscuring the invention.

[0025] 2. Typical Environment

[0026] **FIG. 1** is a block diagram of computer system **100** illustrating a typical environment for implementing the present invention. Computer system **100** may contain one or more processors such as central processing unit (CPU) **110**, random access memory (RAM) **120**, secondary memory **130**, on-chip memory **140**, graphics controller **160**, display unit **170**, output interface **180**, and input interface **190**. All the components except display unit **170** may communicate with each other over communication path **150**, which may contain several buses as is well known in the relevant arts. The components of **FIG. 1** are described below in further detail.

[0027] CPU **110** may execute instructions stored in RAM **120** to provide several features of the present invention. CPU **110** may contain multiple processing units, with each processing unit potentially being designed for a specific task. RAM **120** may receive instructions from secondary memory **130** using communication path **150**. Data may be stored and retrieved from secondary memory **130** during the execution of the instructions.

[0028] Execution of the instructions can provide various features of the present invention. In general, the instructions may represent either user applications or system software. The system software is implemented to provide stack of streams for data transfers (requested by user applications). In addition, the system software enables the user application to control the manner in which memory (in RAM **120**) is allocated.

[0029] On-chip memory **540** may further enhance the performance of computer system **100** as on-chip memory **540** can be implemented along with CPU **110** in a single integrated circuit or tightly cooperating integrated circuits. A stack of streams provided in accordance with an aspect of the present invention may be used to retrieve data from external devices through output interface **190**.

[0030] Various features provided by appropriate implementation of the system software and the manner in which user application may use the feature are described below. The implementation of the system software and user applications will be apparent to one skilled in the relevant arts based on the disclosure provided herein.

[0031] Graphics controller **160** generates display signals (e.g., in RGB format) to display unit **170** based on data/instructions received from CPU **110**. Display unit **170** contains a display screen to display the images defined by the display signals. Input interface **190** may correspond to a key-board and/or mouse, and generally enables a user to provide inputs. Display unit **170** and input interface **190** may together enable a user to develop various user applications on computer system **100**.

[0032] Secondary memory **130** may contain hard drive **135**, flash memory **136** and removable storage drive **137**. Flash memory **136** (and/or hard drive **135**) may store the software instructions (including both system software and user applications) and data, which enable computer system **100** to provide several features in accordance with the present invention.

[0033] Some or all of the data and instructions may be provided on removable storage unit **140**, and the data and instructions may be read and provided by removable storage drive **137** to CPU **110**. Floppy drive, magnetic tape drive, CD-ROM drive, DVD Drive, Flash memory, removable memory chip (PCMCIA Card, EPROM) are examples of such removable storage drive **137**.

[0034] Removable storage unit **140** may be implemented using medium and storage format compatible with removable storage drive **137** such that removable storage drive **137** can read the data and instructions. Thus, removable storage unit **140** includes a computer usable storage medium having stored therein computer software and/or data. An embodiment of the present invention is implemented using software running (that is, executing) in computer system **100**. In this document, the term "computer program product" is used to generally refer to removable storage unit **140** or hard disk installed in hard drive **135**. These computer program products are means for providing software to computer system **100**.

[0035] CPU **110** may retrieve the software instructions, and execute the instructions to provide various features described below in further detail.

[0036] 3. Stack of Streams

[0037] **FIG. 2A** is diagram illustrating the manner in which a stack of streams may be used to efficiently provide data to an application. For illustration, it will be assumed that an application executing on CPU **110** requires data available from a data source (such as hard drive **135**) to be transferred to on-chip memory **140** during execution of a user application. The manner in which system software provided in accordance with an aspect of the present invention can be used by the user application to retrieve the data is described below.

[0038] Broadly, device driver **250** provides a potentially blocking interface to retrieve data from hard drive **135**, and may be implemented in a known way. It should be understood that data may be retrieved from external sources such as another computer over a network interface, and such

implementations are also contemplated to be within the scope and spirit of several aspects of the present invention.

[0039] Second stream **240** transfers the data retrieved by device driver **250** into random access memory (RAM) **120** using a non-blocking approach. For illustration, software may initiate the transfer of data and attend to other tasks while the transfer is occurring. Similarly, first stream **230** interfaces with second stream **240** in a non-blocking manner to retrieve the data from RAM **120** to on-chip memory **140**.

[0040] The user application may use the non-blocking interface provided by first stream **230** to have the data retrieved. By separating the data transfer into two streams (**230** and **240**), first stream **230** may be implemented using potentially pre-existing software code for second stream **240**.

[0041] It should be understood that system software provided in accordance with an aspect of the present invention enables user applications to cause the data to be retrieved as described above. The manner in which a user application can interface with system software is described below.

[0042] 4. Example Application Software Code

[0043] **FIG. 2B** depicts example software code (transferring data) contained in a user application illustrating the manner in which multiple non-blocking streams can be set up and used for data transfer. Line **260** associates an instance of device driver **250** to the variable DVR. DRIVER routine (commonly referred to as function in the relevant technology arts) can be implemented in a known way in the system software. The statement of line **265** defines (specifies) second stream **240** (STREAM2) (provided as a part of the system software) to use DVR to retrieve data. Similarly, the statement of line **266** defines first stream **230** (STREAM1) to use second stream **240** to retrieve data.

[0044] The statement at line **270** opens (activates) an instance of a stream on the STMDEV1 device. The stream is assigned to the variable (label) STM1. For conciseness, the description is provided with reference to retrieving data. However, the concepts may be applied to writing data as well.

[0045] The issue statement of line **280** initiates non-blocking data transfer using instance STM1. The issue statement requests a number of bytes specified by SIZE, and a pointer to a buffer where the data may be stored is specified by BUFP. As the issue statement is non-blocking, the user application can continue with performing other tasks (as shown by the dots of lines **281-283**).

[0046] The reclaim statement of line **284** is used by the application to start accessing and using the data retrieved by first stream **230**. If all the requested data is not available, the reclaim statement may be blocked until all the data becomes available. The user application may continue processing the data (or with other tasks) once the reclaim returns with the retrieved data.

[0047] Thus, a stack (two or more) non-blocking streams may be used by a user application to efficiently retrieve data. The user application generally needs to be supported by an appropriate implementation of system software. Some implementation considerations in an example system software are described below.

[0048] 5. Implementation of System Software

[0049] In general, the system software needs to be implemented consistent with the syntax and semantics chosen for the various types of statements in the application code. Each statement needs to be parsed to determine various parameters related to the specific task the statement is intended to perform. The system software needs to be determined to execute the corresponding task. The implementation considerations to support the user application code of **FIG. 2B** are described below.

[0050] In an embodiment, the system software may be designed to receive multiple issue statements and a corresponding number of reclaim statements. A first-in-first-out (FIFO) approach may be used to associate each returned reclaim statement (completed data transfer) with the corresponding issue statement. Thus, several issue statements may be pending for each (instance of) stream. All the pending issue statements may be maintained in a queue ("issue queue") for the stream.

[0051] There are at least three different situations when a new issue statement is received by the system software—(1) issue queue is full (i.e., number of maximum allowed outstanding issue statements is exhausted); (2) issue queue is empty; and (3) issue queue is neither empty nor full. Support generally needs to be present for all the three situations in both streams **230** and **240**.

[0052] For situation (1), the new issue statement may not be accepted, and an error may be returned. For situation (2), with respect to streams which rely on lower streams, an issue statement is generated to the stream at the lower layer. On the other hand, when the issue queue is empty in second stream **240**, device driver **250** may be used in a blocking manner to retrieve the data. For situation (3), the issue statement is merely added to the issue queue (in the stream received in).

[0053] In an embodiment, a separate process may be maintained to examine each issue queue, and cause data transfers in the lower streams (or device drivers) to generate issue statements (or use device driver in the case of second stream **240**) to the lower stream corresponding to the issue statements in the issue queue.

[0054] Once the data transfer corresponding to each issue statement is complete, a return queue is maintained to provide the data to the corresponding reclaim statements originating from the higher layer. In an implementation, an application/layer is blocked if the data corresponding to a reclaim statement is not available. If the data is available, the reclaim statement returns immediately and the data (in the buffer) is made available for further processing.

[0055] The system software may coordinate the data transfers between various layers as described above. The coordination also typically entails buffer management, which may be performed as follows.

[0056] 6. Buffer Management

[0057] A combination of approaches may be used in buffer management to efficiently provide data transfers to user applications. Some example approaches are described below.

[0058] Whenever permitted by the data transfers, the same buffer may be used to transfer data across multiple layers

such that the number of copies to different buffers is minimized. Typically, a pointer to a buffer is transferred across layers to provide the data in the buffer. Such an approach may be referred to as "zero buffer copy". Such an approach is typically possible when the buffers maintained by the layers are on the same storage medium. So instead of copying a buffer into another buffer as it is passed from layer to layer, a reference to the same buffer can be passed.

[**0059**] In addition, multiple buffers may be used to retrieve data related to several issue statements in parallel. By using multiple buffers, the application may operate on one buffer and the 'system software' can process other buffers in parallel. This is commonly referred to double-buffering. As a result, the reclaim statements from the higher layers may return quickly.

[**0060**] For each of the buffers used to store data, the corresponding memory space may need to be allocated such that the data intended for consumption for one entity (application/layer) is not unintentionally overwritten by some other entity. An example approach for memory management is described below.

[**0061**] 7. Memory Management

[**0062**] According to another aspect of the present invention, the user applications are provided greater control in managing the memory space available in RAM **120** and/or on-chip memory **140**. The manner in which an application may be provided such control in securing the desired buffer space for the streaming calls is described with reference to example user application code contained in **FIG. 3**.

[**0063**] In the statements of lines **310-313** (of the user application code), a user may define the allocation logic (my-alloc) to be employed in allocating memory space. In one embodiment, the entire memory space is partitioned into multiple sub-spaces, and each sub-space is assigned a base manager. My-alloc may thus be implemented to interface with the base managers for any designed memory space.

[**0064**] In the statements of lines **320-323**, a user may define the release logic (my-free) to free any prior allocated memory space. In general, the release logic needs to be implemented consistent with the allocation logic. For example, with reference DSPBIOS environment developed by Texas Instruments, the assignee of the subject patent application, assume that the stream buffers are to be assigned in a memory segment (sub-space) MEM**1**. In this scenario, my-alloc and my-free could be written as follows:

[**0065**] void *my-alloc (int size) { int align=4; return MEM_alloc (MEM**1**, size, align);}

[**0066**] void my-free (void *ptr, int size) {return MEM_free (MEM**1**, ptr, size);}

[**0067**] The convention for user application code and system software needs to enable the application software to use the user defined allocation and release logics. In addition, the system software needs to be implemented to provide the user applications the interface to access the basic primitives (MEM-alloc and MEM_free above) which provide the low level memory management. The user application code can then be written to provide custom higher level memory management features as desired by a programmer.

[**0068**] Continuing with reference to **FIG. 3**, in the statements of lines **330-333**, my-alloc and my-free are defined to be part of a type of data structure (mem-mgr). Variable my-mem-mgr is set equal to an instance of the data-structure mem-mgr. In the statement of line **350**, the variable my-mem-mgr is associated with STMDEV**3** stream device. In statement **351**, a stream labeled stm**3** is opened for write access (using 'w').

[**0069**] In the statement of line **360**, an alloc statement is issued to request a-number of bytes (size 3) for the memory space. The alloc statement is designed to use a default logic if none (i.e., no my-alloc and my-free) is specified in the statement of line **350**. Once the memory space is allocated, the user application may write data into the buffer (buf**3**).

[**0070**] In statement of line **370**, an issue statement is generated from the application to cause the data to be written to a device (for example using device driver **250**). In statement of line **380**, a reclaim statement is generated. The buffer is then released in the statement of line **390**.

[**0071**] Thus, using the approaches of above, the present invention allows data to be provided to user applications. Some of the implementation details of an embodiment of the system software are described in the Appendix in the form of pseudo-code well known in the relevant arts. An aspect of the invention allows efficient data transfer by using a stack of non-blocking streams. Another aspect of the present invention allows for flexible memory management by providing the user applications to control the specific areas of a memory.

[**0072**] 8. Conclusion

[**0073**] While various embodiments of the present invention have been described above, it should be understood that they have been presented by way of example only, and not limitation. Thus, the breadth and scope of the present invention should not be limited by any of the above described exemplary embodiments, but should be defined only in accordance with the following claims and their equivalents.

Appendix

```
/*****************************************************************************
Initialization
*****************************************************************************/

status initPhyDriver(driver specific parameters)
{

1. Allocate/reserve any resources needed by the driver.
2. Initialize the physical driver, and prepare it to read or write data.
3. Initialize a queue to maintain data transfer requests.
}

status initStreamDevice(stream_device name, underlying entity, memory
management procedures)
{

The underlying entity may be one of:
     1. A physical driver. In this situation, this stream device interfaces
with the hardware.
     2. Another stream device. Streams opened on such devices will
          effectively stacked above another stream layer. 1. Allocate
          resources necessary to maintain the device. 2. Add an entry to
          the global device list with the passed name. 3. Some devices
          may have to store data as 'files'. In this the related
structures should be allocated and initialized.
}
```

```
/*****************************************************************************
Stream Interface
*****************************************************************************/

stream streamOpen(stream_device name, stream name, attrs)
{
 attrs contains:
 1.  The mode i.e.  read or write.  2.  The initial number of buffers.  3.
The size of each buffer.  1.  Find the stream_device name in the list of
devices, and get the handle
 to the device.  2.  Find the stream name in the device table.  3.  If the
stream is to be opened in read mode, and the name does not exist, flag an
error.  4.  If the stream is to be opened in write mode, and the name exists,
 overwrite the existing entry.  5.  Allocate memory to store the stream
structure.  6.  Initialize the structure with the device id, the stream id,
and the mode.  7.  Initialize any other device specific contents of the
structure.  8.  Initialize the 'from' and 'to' queues, to accomadate the max
number of
 buffers specified in 'attrs'.  9.  If this is a bottom level stream, ensure
that the underlying drivers are
 opened.
}

status streamIssue(stream, buffer, size)
{
 1.  If the number of outstanding buffers is the max specified, flag an
 error.  2.  Form a buffer request, and add it to the 'to' queue, and
increment the
```

```
number of outstanding buffers. 3. Issue the request to the next layer.
This can either be a physical
driver, or another stream layer. In case it is a physical driver, this
involves adding the request to the 'driver' queue.
}

status streamReclaim(stream, buffer reference, size)
{

1. If the number of outstanding buffers is zero, flag an error. 2. If
there are some entries in the 'from' queue, pass the buffer back to
the application, and remove it from the queue. In this case, the
application does not encounter any delay. 3. If there are no entries in the
'from' queue, reclaim a buffer from the
next layer, and pass the returned buffer back to the application. Reclaiming
it from the next layer may need the application to be
blocked. Remove the first entry in the 'to' queue. 4. Decrement the number
of outstanding buffers.
}

/* In an embodiment, this interface is not called by the application but
within the system. */
status streamNotify(stream, buffer, size)
{

1. If the buffer does not match with the first entry in the 'to' queue,
flag an error. 2. Remove the first entry from the 'to' queue, and push it
to the end of
the 'from' queue. 3. Notify the previous layer.
```

```
}

buffer streamAlloc(stream, size)
{

 1.  If the buffer size is not as expected, flag an error.  2.  Allocate the
required memory using the application specific memory
 manager.  3.  If necessary, initialize the memory.
}

status streamFree(stream, buffer, size)
{

 1.  Free the memory using the applicaton specific memory manager.
}

status streamClose(stream)
{

 1.  Free all resources used by the 'stream'.
}
```

```
/*****************************************************************************
Background thread.
This is a low priority thread which runs in the background, and is responsible
for executing stream requests when the processor is idle.
```

```
**************************************************************************/

status streamThread()

{

1. Loop over all the drivers in the system, checking for any data

 transfer requests.  2.  Execute any pending data transfer requests.  3.

Notify the previous stream layer on completion, and continue the loop.

}
```

What is claimed is:

1. A method of transferring data from a data source to a user application, said user application being executed in a system, said method comprising:

(a) enabling a plurality of non-blocking streams to be set up between said user application and said data source; and

(b) enabling said user application to generate a statement to initiate the transfer of data from said data source using said plurality of non-blocking streams.

2. The method of claim 1, wherein said plurality of non-blocking streams comprise a stack of non-blocking streams.

3. The method of claim 2, wherein (a) comprises including a plurality of statements in said user application which cause said stack of streams to be specified and activated.

4. The method of claim 3, further comprising enabling a blocking transfer of data between a last one of said stack of non-blocking streams and said data source.

5. The method of claim 4, wherein said data source comprises one of a peripherals within said system or a source external to said system, and a first one of said stack of non-blocking streams interfaces with a driver retrieving data from said data source and a second one of said stack of non-blocking streams interfaces with said first one of said stack of non-blocking streams to provide said data to said user application.

6. The method of claim 2, further comprising using a buffer to transfer data between any two of said stack of non-blocking streams.

7. The method of claim 6, wherein said buffer is implemented using a memory space in a memory, said method further comprises:

(c) enabling said user application to control allocation of the specific portions of said memory as said memory space.

8. The method of claim 7, wherein (c) comprises including a plurality of statements representing a logic to allocate said memory space.

9. A computer readable medium carrying one or more sequences of instructions for causing transfer of data from a data source to a user application, said user application being executed in a system, wherein execution of said one or more sequences of instructions by one or more processors contained in said system causes said one or more processors to perform the action of:

(a) enabling a plurality of non-blocking streams to be set up between said user application and said data source; and

(b) enabling said user application to generate a statement to initiate the transfer of data from said data source using said plurality of non-blocking streams.

10. The computer readable medium of claim 9, wherein said plurality of non-blocking streams comprise a stack of non-blocking streams.

11. The computer readable medium of claim 2, wherein (a) comprises including a plurality of statements in said user application which cause said stack of streams to be specified and activated.

12. The computer readable medium of claim 11, further comprising enabling a blocking transfer of data between a last one of said stack of non-blocking streams and said data source.

13. The computer readable medium of claim 12, wherein said data source comprises one of a peripherals within said system or a source external to said system, and a first one of said stack of non-blocking streams interfaces with a driver retrieving data from said data source and a second one of said stack of non-blocking streams interfaces with said first one of said stack of non-blocking streams to provide said data to said user application.

14. The computer readable medium of claim 10, further comprising using a buffer to transfer data between any two of said stack of non-blocking streams.

15. The computer readable medium of claim 14, wherein said buffer is implemented using a memory space in a memory, further comprises:

(c) enabling said user application to control allocation of the specific portions of said memory as said memory space.

16. The computer readable medium of claim 15, wherein (c) comprises including a plurality of statements representing a logic to allocate said memory space.

17. A system for executing a user application which requires data from a data source, said system comprising:

means for enabling a plurality of non-blocking streams to be set up between said user application and said data source; and

means for enabling said user application to generate a statement to initiate the transfer of data from said data source using said plurality of non-blocking streams.

18. The system of claim 17, wherein said plurality of non-blocking streams comprise a stack of non-blocking streams.

19. The system of claim 18, further comprising means for enabling a blocking transfer of data between a last one of said stack of non-blocking streams and said data source.

20. The system of claim 19, further comprises a memory means providing a buffer means to transfer data between any two of said stack of non-blocking streams, wherein said buffer is implemented using a memory space in a memory, said system further comprises: means for enabling said user application to control allocation of the specific portions of said memory as said memory space.

* * * * *